Slide-1

**Explain "Operating system is interrupt driven"**

An operating system is described as an "interrupt-driven software" because basically everything that goes on software-wise is some sort of interrupt to the OS. An interrupt driven device sends an interrupt request to the computer, which is then serviced by Internet service routine (ISR). To effectively manage many processes. the core of operating system makes use of what is known as interrupts. This is a signal to a processor indicating that an asynchronous event has occurred. that's why operating system is interrupt driven.

Slide-2

**1. Explain different types of storage structures (main memory /secondary memory)**

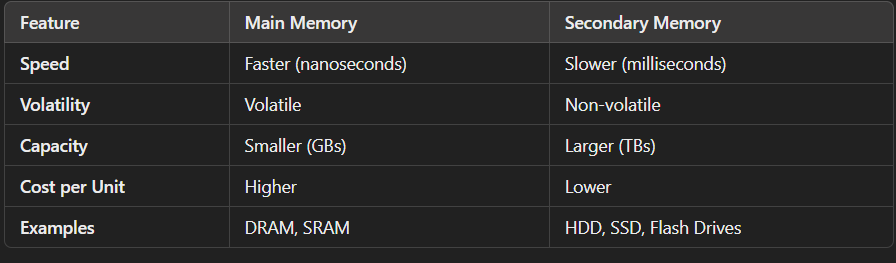
**Answer:** Computer storage is divided into main and secondary memory, differing in speed, volatility, and capacity.

**Main Memory**

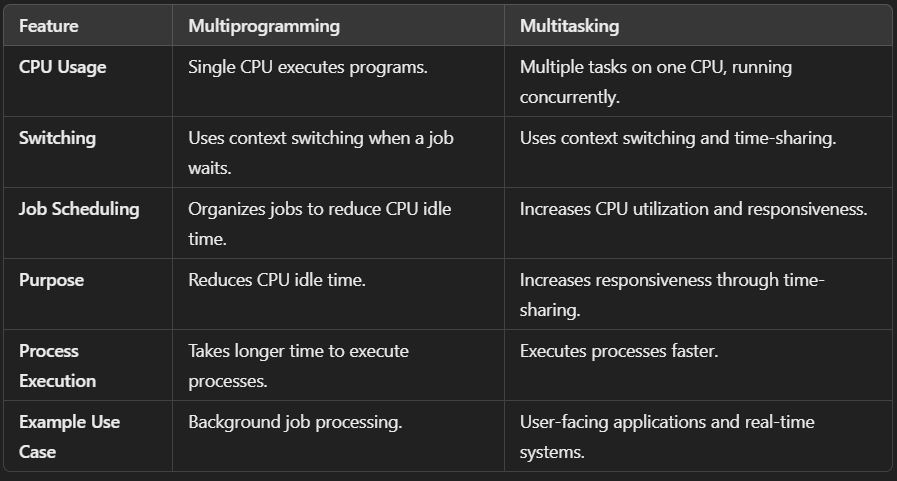
* **Random Access:** Direct CPU access.
* **Volatile:** Loses data when powered off.
* **RAM (e.g., DRAM):** Temporary, high-speed storage for active processes.

**Secondary Memory**

* **Non-Volatile:** Retains data without power.
* **HDD:** Slower, high-capacity storage on magnetic platters.
* **NVM (e.g., SSD, NVMe):** Faster, durable storage with no moving parts.



**2. Explain Multiprogramming vs Multitasking**

**Answer:** 

**3. Explain user mode and kernel mode of OS. how it works with diagram**

**Answer:**

1. **User Mode:**
   * Restricted access to system resources.
   * Runs user applications.
   * Relies on system calls to request kernel services.
2. **Kernel Mode:**
   * Full access to hardware and system resources.
   * Executes OS tasks like memory and device management.
   * Can perform privileged operations.

**Working:**

* The CPU switches between modes using a **mode bit**:
  + **User Mode (1):** For applications.
  + **Kernel Mode (0):** For OS operations.
* System calls transfer control from user mode to kernel mode and back after execution.

Slide 3

**1. Explain Operating System Services +OS architecture diagram**

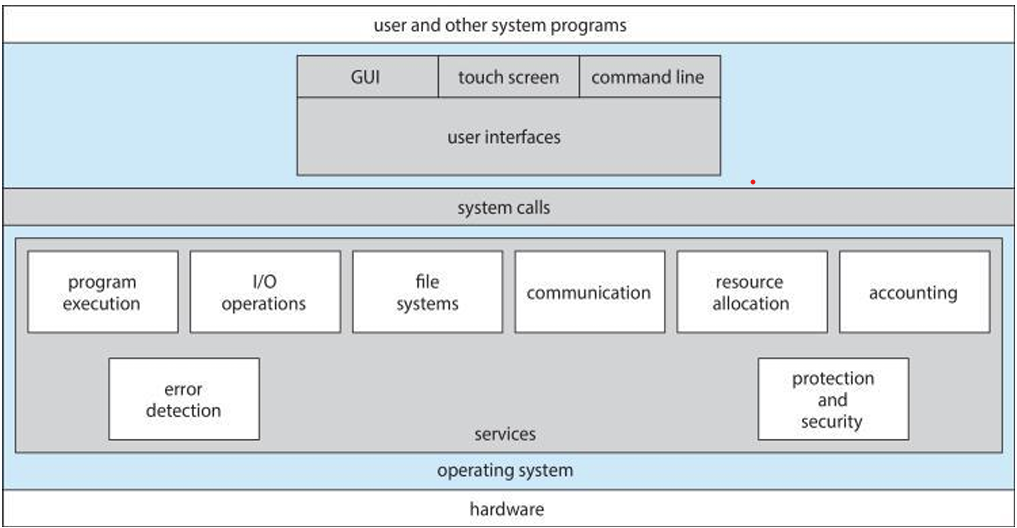
**Answer:**

**Operating System Services:**

1. **User Interface (UI):** Allows user interaction (CLI, GUI, touch, batch).
2. **Program Execution:** Loads, runs, and terminates programs.
3. **I/O Operations:** Manages input/output for files and devices.
4. **File-System Manipulation:** Handles file creation, deletion, reading, writing, and permissions.
5. **Communications:** Lets programs share data.
6. **Error Detection:** OS needs to be constantly aware of possible errors.

**Resource Management:**

1. **Resource Allocation:** Allocates CPU, memory, and devices to users.
2. **Logging:** Records resource usage.
3. **Protection and Security:** Controls access to system resources and protects from unauthorized access.



**2. What is system call? Explain types of System Call**

**Answer:**

### A system call lets a program request services from the OS, like managing hardware or performing I/O.

### ****Types of System Calls:****

1. **Process Control:** Manage processes (create, terminate, load, execute), memory allocation, debugging, event waiting.
2. **File Management:** Handle files (create, delete, open, read, write).
3. **Device Management:** Manage devices (request, release, read, write).
4. **Information Maintenance:** Set/get system time, data, and attributes.
5. **Communications:** Handle connections, message passing, shared memory.
6. **Protection:** Manage resource access and permissions.

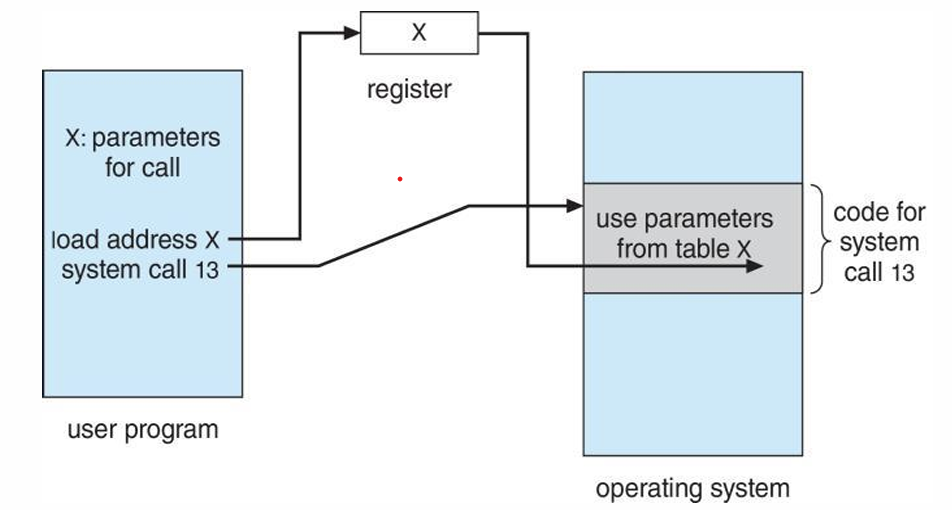
### 3. System Call parameter Passing with diagram

### ****System Call Parameter Passing:****

Parameters can be passed in three ways:

1. **Registers:** Directly in CPU registers (simplest).
2. **Memory Block:** Stored in memory, address passed in a register (used by Linux, Solaris).
3. **Stack:** Pushed onto the stack by the program, popped by the OS.

Block and stack methods allow for more parameters and larger data.

****

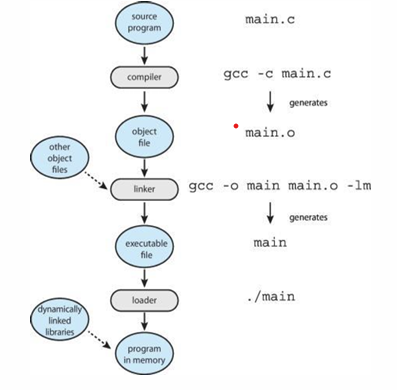
**4. Explain Linkers and Loaders with diagram**

**Answer:**

**Linker and Loader:**

* **Linker:** Combines different parts of a program and links necessary libraries, setting correct addresses.
* **Loader:** Loads the program from storage into memory to run it, making sure everything is in place.

Modern systems load libraries only when needed and share them across programs.

****

**5. Why Applications are Operating system specific? Explain**

**Answer:**

Applications are OS-specific because each OS has unique system calls, file formats, and libraries. Here's why:

* **System Calls:** Each OS has its own system calls for hardware, memory, and file interaction.
* **File Formats:** Apps are compiled for OS-specific formats, making them incompatible across systems.
* **Multi-OS Applications:** Apps in interpreted languages (e.g., Python) or with a VM (e.g., Java) can run on multiple OSes.
* **Compiling for Each OS:** Apps like C must be compiled separately for each OS.
* **ABI:** Defines how binary code interacts with the OS and hardware for compatibility.

**6. Explain different type of OS structures like MS DOS, LINUX, Layered, Microkernel with diagram**

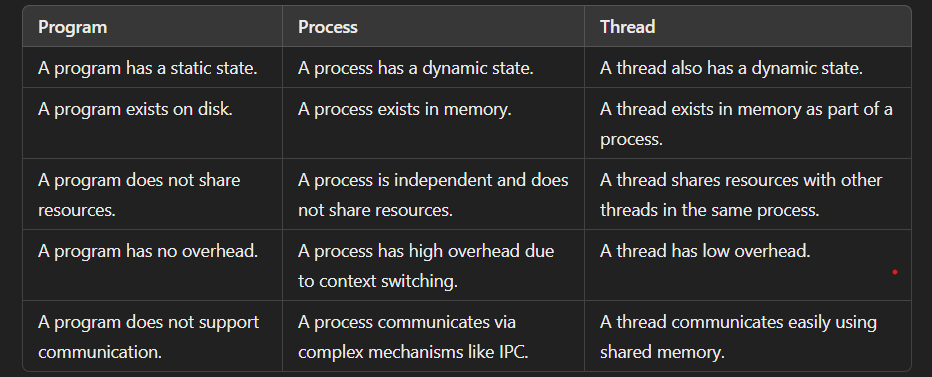
**Answer:**

**Types of OS Structures:**

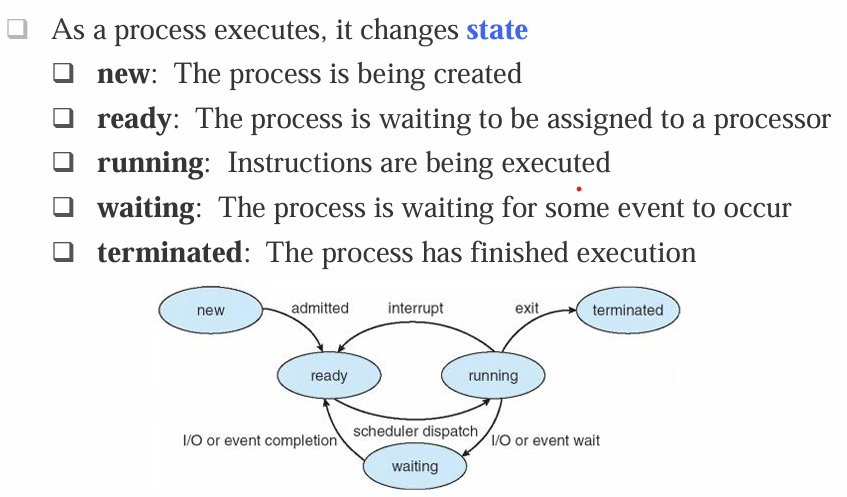
1. **MS-DOS:**
   * Simple, single-user, command-line OS.
   * No multitasking or memory protection; monolithic structure.
2. **Linux:**
   * Monolithic, multi-user, multitasking OS.
   * Single large kernel with modular capabilities via loadable modules.
3. **Layered OS:**
   * OS divided into layers, each with specific tasks.
   * Enhances modularity and maintainability.
4. **Microkernel:**
   * Minimal kernel with basic functions (e.g., resource management).
   * Other services run in user space, improving security and stability.

Slide 4

Explain difference among program, process and thread



Process State diagram with explanation.



**Explain Process Scheduling with diagram (job, read, device queues)**

Process Scheduling is how the operating system decides which process runs on the CPU and when.

* + Maximize CPU use quickly switch processes for time sharing
  + A process gives up the CPU in two cases:
    - I/O Request
    - After a set time (N units), apply by a timer.
  + Once a process gives up the CPU it is added to the “ready queue”
  + The **scheduler** selects the next process from the ready queue to run on the CPU

**Scheduling Queues**

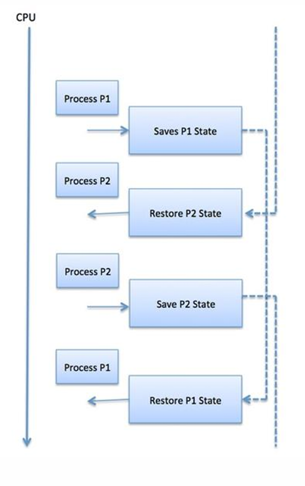
* **Job Queue**: Contains all processes in the system.
* **Ready Queue**: Processes in memory, ready to execute.
* **Device Queues**: Processes waiting for specific I/O devices.



**Explain context switching with diagram**

When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch

* Context-switch time is pure overhead; the system does no useful work while switching.
* The more complex the OS and the PCB ➔ the longer the context switch

****

**Different types of schedulers.**

**Short-term scheduler (CPU scheduler)**: Chooses the next process to run and assigns the CPU.

 May be the only scheduler in some systems.

 Runs frequently (every few milliseconds) and must be fast.

**Long-term scheduler (Job scheduler)**: Decides which processes enter the ready queue.

 Runs infrequently (every few seconds or minutes) and can be slow.

 Controls the level of multiprogramming.

**Zombie process vs orphan process**

❑ If no parent waiting (did not invoke wait()) process is **zombie[child work is done but waiting for parent to complete]**

* Status table yes but running no but parent process need to call exit() system

❑ If parent terminated without invoking wait, process is **orphan**

* Running yes but parent running no

**Explain shared memory vs message passing**

**Shared Memory Systems:**

* A region of memory that is shared by cooperating processes.
* An area of memory shared among the processes that wish to communicate

**Message Passing Systems:**

* A mechanism for processes to communicate and synchronize by exchanging messages.
* Without resorting to shared variables

IPC facility provides two operations:

❑ send(message)

❑ receive(message)

**Blocking and Non-blocking schemes**

 **Blocking (Synchronous)**:

* **Blocking send**: Sender is blocked until the message is received.
* **Blocking receive**: Receiver is blocked until a message is available.

 **Non-blocking (Asynchronous)**:

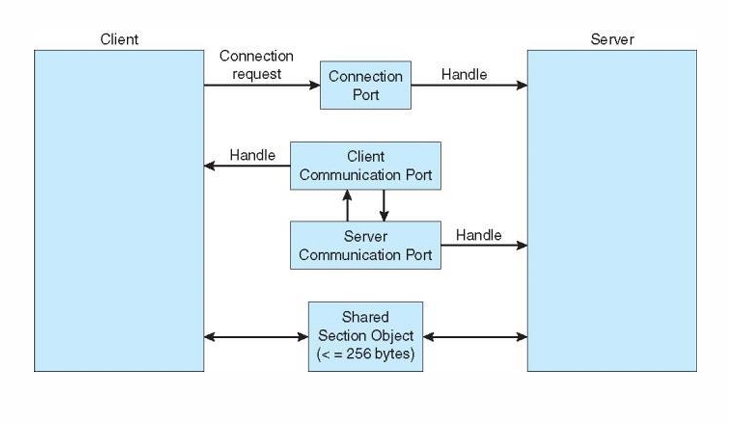
* **Non-blocking send**: Sender sends the message and continues.
* **Non-blocking receive**: Receiver gets either a valid message or a null message.

 **Rendezvous**: Occurs when both send and receive are blocking, requiring synchronization between sender and receiver.

**Explain windows LPE with diagram (last slide diagram) \*\*\*(most important)**

Local Privilege Escalation (LPE) allows attackers to gain higher privileges on a compromised system. In Windows, it often stems from flaws in IPC, access control, or insecure object handling.

**Diagram:**



**Diagram Explanation:**

1. **Client and Server Relationship:**
   * Client requests a connection via a **Connection Port**.
   * Server provides a **Handle** for further communication.
2. **Communication Ports:**
   * **Client Port**: Sends/receives data.
   * **Server Port**: Processes client requests.
3. **Shared Section Object:**
   * Shared memory (≤256 bytes) enables quick data exchange between client and server.

**Slide-5**

**Benefits of multithreading**

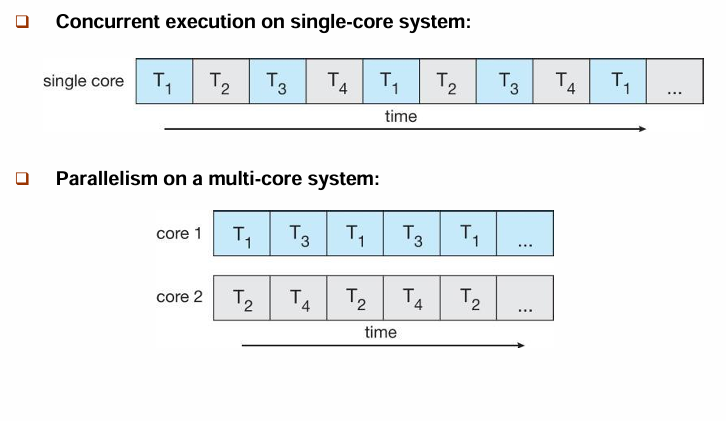
 **Responsiveness**: Keeps working even if part of the process is blocked, useful for user interfaces.

 **Resource Sharing**: Threads share the same resources, easier than other methods like shared memory.

 **Economy**: Creating threads is cheaper than creating processes, and switching between threads uses less power.

 **Scalability**: Can use multiple processor cores to improve performance.

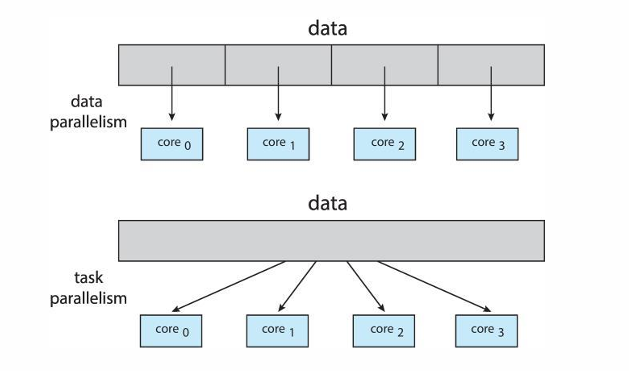
**Explain concurrency vs Parallelism**

****

**Data parallelism vs task parallelism**

 **Data Parallelism**: Splits the same data into parts and processes them on multiple cores at the same time.

 **Task Parallelism**: Splits tasks (or threads) and runs them on different cores.

****

**Explain 3 multithreading models with diagram**